Università degli Studi di Salerno
Corso di Ingegneria del Software

CarFix
Object Design Document
Versione 1.0

Data: 16/12/2024

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
Felice Chirico	0512108085
Carmine Nunziata	0512116290

Scritto da:	Nunziata Carmine

Revision History

Data	Versione	Descrizione	Autore
16/12/2024	1.0	Prima stesura del documento	Nunziata Carmine

Indice

1. Introduzione
    1.1. Object Design Trade-Offs
    1.2. Linee guida per la documentazione d'interfaccia
    1.3. Definizioni, acronimi e abbreviazioni
        1.3.1. Definizioni
        1.3.2. Acronimi
    1.4. Riferimenti
    1.5. Packages (Revisione)
        1.5.1. View
            1.5.1.1. User
            1.5.1.2. CarFix
            1.5.1.3. Order
            1.5.1.4. Review
            1.5.1.5. Catalog

1. Introduzione

1.1 Object Design trade-offs

- **Leggibilità vs Tempo**
  Il codice appartenente all'intero progetto dovrà essere ben documentato ai fini di test e/o eventuali modifiche. Per raggiungere queste aspettative si ricorrerà all'uso di commenti, annotazioni o qualsiasi altra tipologia di documentazione per semplificare la comprensione e la lettura del codice data la possibile presenza di funzioni complesse o articolate. Per finalizzare tale obiettivo sarà necessario tuttavia dedicare maggior tempo allo sviluppo del progetto.

- **Riuso vs Semplicità**
  Oltre a maggiore semplicità, il progetto CarFix mira al recupero di tempo utilizzato per garantire leggibilità con l'uso di funzioni o componenti riutilizzabili e quanto più generici ed adattabili possibili. Altro punto a favore, oltre al recupero del tempo, è quello di ottenere un ambiente più semplice su cui operare per mansioni semplici o complesse di manutenzione. Al contrario, sebbene si velocizzi il processo complessivo di realizzazione, si otterrà maggiore complessità di design vista la necessità e lo sviluppo di componenti flessibili e riutilizzabili in ogni punto del codice.

- **Sicurezza vs Efficienza**
  Trattandosi di un e-commerce è scontato che l'utente si aspetti che la piattaforma sia sicura sul trattamento dei dati sensibili, tuttavia dato il tempo ridotto per la consegna del prodotto finito, sarà implementata una sicurezza sia sulle password degli utenti (per prevenire furti di account) e sulla memorizzazione ed uso di dati bancari.

1.2 Linee guida per la documentazione d'interfaccia

Sono state stabilite le seguenti linee guida per la scrittura del codice:

- ➔ **Classi**
  - ◆ Ogni classe avrà, come iniziale del proprio nome, una lettera maiuscola;
  - ◆ Il nome attribuito ad ogni classe espliciterà informazioni utili relative al proprio scopo e sarà solo e soltanto composto da una parola.

- ➔ **Interfacce**
  - ◆ Il nome delle interfacce sarà semplice e con l'utilizzo della tecnica camelCase come per tutte le altre componenti per permettere la realizzazione di contratti chiari e comprensibili, evitando complessità non necessarie.
  - ◆ Come per le altre componenti, saranno utilizzate nomenclature mirate a descrivere esattamente il ruolo dell'interfaccia.

- ➔ **Pagine**
  - ◆ Ogni file sorgente conterrà solo e soltanto una classe e vedrà uno schema ben preciso: definizione del package di appartenenza, import di tutte le librerie necessarie, documentazione (per garantire leggibilità come esplicato in precedenza, utilizzando Javadoc) e il codice effettivo.

- ➔ **Metodi**
  - ◆ Per il nome di ogni metodo sarà utilizzata la tecnica camelCase;
  - ◆ Il nome attribuito ad ogni metodo sarà mirato all'esplicazione di ciò che permette di fare e, dunque, facilitare la comprensione e la manutenzione del codice visto che ogni nome indicherà l'azione che svolgerà quella parte di codice;
  - ◆ Saranno utilizzati metodi getter e setter per ogni classe.

- ➔ **Errori**
  - ◆ Il nome di eventuali eccezioni programmate seguirà la tecnica usata per le Classi;
  - ◆ Qualsiasi eccezione sarà gestita utilizzando il costrutto try-catch, permettendo di gestirle al meglio senza causare interruzioni improvvise del codice.

- ➔ **Variabili**
  - ◆ Per il nome di ogni variabile sarà utilizzata la tecnica camelCase;
  - ◆ Il nome attribuito ad ogni variabile indicherà esattamente ciò che essa rappresenta all'interno dell'esecuzione;

1.3 Definizioni, acronimi e abbreviazioni

1.3.1 Definizioni

TERMINE	DEFINIZIONE
JAVADOC	*Strumento per la realizzazione di documentazione specifico del linguaggio Java.*
VIEW	*Responsabile della rappresentazione grafica di dati ed interfaccia utente.*
SERVLET	*Oggetti scritti con sintassi Java in grado di operare all'interno di un server web.*
CONTEXT	*Informazioni condivise che definiscono lo stato di un'entità.*

1.3.2 Acronimi

ACRONIMO	RIFERIMENTO
RAD	*Requirement Analysis Document*
SDD	*System Design Document*
ODD	*Object Design Document*
UML	*"Unified Modeling Language": linguaggio grafico per la modellazione di sistemi software.*
OCL	*"Object Constraint Language": linguaggio che specifica restrizioni e vincoli dei modelli UML.*
OOP	*"Object-Oriented Programming": modello di programmazione che consente di definire oggetti in grado di interagire gli uni con gli altri.*
JDBC	*"Java DataBase Connectivity": driver che consente l'implementazione di interfacce Java per accedere e gestire un DataBase.*
JSP	*"Java Server Page": linguaggio in grado di creare pagine web dinamiche usando la sintassi Java.*
DAO	*"Data Access Object": oggetti utilizzati per facilitare l'accesso ai dati a database relazionali.*

1.4 Riferimenti

Il progetto mira ad ampliare e semplificare il mercato dei pezzi di ricambio delle autovetture, rendendo dunque molto più facile anche al più inesperto distinguere ed acquistare le varie tipologie di ricambio. Punto cardine e d'ispirazione è la nota piattaforma Autodoc che, tuttavia, molto spesso risulta poco intuitiva e dunque si mira alla realizzazione di una piattaforma altresì completa e molto più semplice ed immediata per i clienti.

- ➔ Riferimento a documenti
  - ◆ **RAD**;
  - ◆ **SDD**
- ➔ Riferimenti tecnici
  - ◆ **OOP**;
  - ◆ Front-End
    - ● **JSP**
    - ● HTML, CSS, XML
  - ◆ Back-End
    - ● **JDBC**
    - ● **Javadoc**
    - ● **Context**
    - ● **View**
    - ● **DAO**

1.5 Packages (revisione prevista)

E’ possibile definire i seguenti package:

- ➔ view
- ➔ userManagement
- ➔ orderManagement
- ➔ productManagement

1.5.1 View

1.5.1.1 User

Il sottopacchetto “`User`” contiene tutte le servlet adibite per le funzioni della gestione degli account e di autenticazione del sistema.

`view.User`	
CLASSE	**DESCRIZIONE**
EditProfile	*Servlet per la modifica del profilo utente*
LoginServlet	*Servlet per la gestione del login alla piattaforma*
SignupServlet	*Servlet per la gestione della registrazione alla piattaforma.*
LogoutServlet	*Servlet per la disconnessione dalla piattaforma.*
MyOrderServlet	*Servlet per la gestione degli ordini effettuati.*

1.5.1.2 CarFix

Il sottopacchetto "`CarFix`" contiene le classi Java per la connessione al Database e alla gestione dei Context.

`view.CarFix`	
CLASSE	**DESCRIZIONE**
DbManager	*Classe che consente la connessione al database e l'interazione coi dati usando la libreria JDBC.*
MainContext	*Classe per la gestione dei Context: inizializzazione e distruzione.*

1.5.1.3 Order

Il sottopacchetto "`Order`" contiene le servlet adibite alle funzioni di acquisto della piattaforma.

`view.Order`	
CLASSE	**DESCRIZIONE**
MakeOrderServlet	*Servlet che permette la creazione di un ordine.*
CartServlet	*Servlet che permette la gestione del carrello.*

1.5.1.4 Review

Il sottopacchetto "`Review`" contiene le servlet adibite alle funzioni di recensioni.

`view.Review`	
CLASSE	**DESCRIZIONE**
ReviewManager	*Servlet che permette la gestione di una recensione.*

1.5.1.5 Catalog

Il sottopacchetto "`Catalog`" contiene le servlet adibite alle funzioni per la gestione del catalogo.

`view.Catalog`	
CLASSE	**DESCRIZIONE**
AddRicambioServlet	*Servlet che permette l'aggiunta di un ricambio al catalogo*
RemoveRicambioServlet	*Servlet che permette la rimozione di un ricambio dal catalogo*
UpdateRicambioServlet	*Servlet che permette di modificare le informazioni di un ricambio*